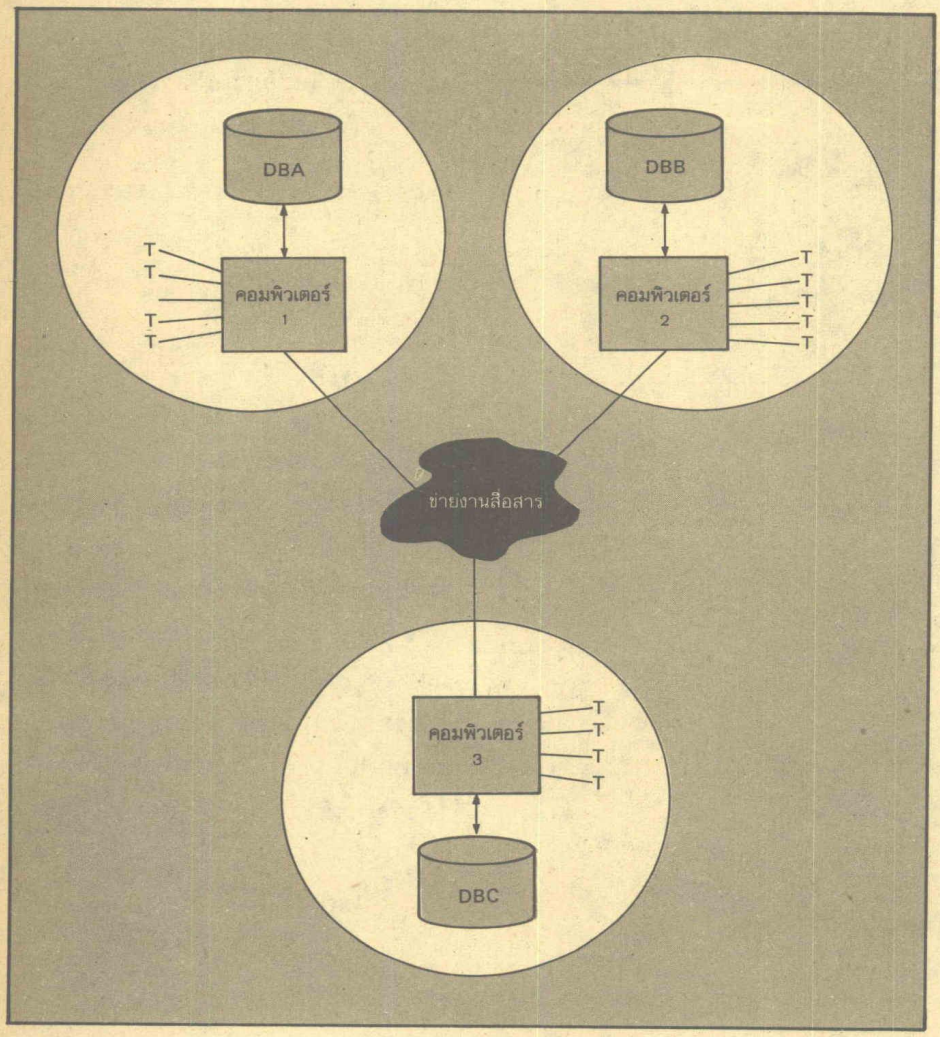
ปัญหาน่าสนใจเกี่ยวกับฐานข้อมูลแบบกระจาย

ครรชิต มาลัยวงศ์

"ช่วยเลือกห้องทำงานที่เงียบๆ ให้ผมหน่อย ผมจะมาถึงกรุงเทพฯ 15 มกราคมนี้" นั่นเป็นข้อความตอนหนึ่งในโทรสารที่ได้รับจากหนุ่มชาวนอร์เวคนหนึ่งชื่อ มอดส์ นีการ์ด ผู้เป็นนักวิจัยอยู่ที่สถาบันซินเทฟ ในเมืองทรอนด์ไฮม์ ตอนกลางของประเทศและเป็นที่ตั้งสถาบันเทคโนโลยีที่สำคัญแห่งหนึ่ง



รูปที่ 1 ฐานข้อมูลแบบกระจายออกไปติดตั้งหลายแห่ง

ผมเคยพบนีการ์ดมาแล้วเมื่อครั้งเดินทางไปจีบให้รัฐบาลนอร์เวออกเงินช่วยเหลือสถาบันที่ยากจนอย่างเอไอที ในครั้งนั้นผมคงได้พบชาวไวกิ้งนับร้อยจนจำไม่ได้ว่าใครเป็นใคร ดังนั้นเมื่อผมได้รับโทรสารจึงเกิดความรู้สึกว่า อาจารย์ใหม่คนนี้คงจะเก็บตัว สมถะเงียบๆ ไม่ต้องการพบกับความพลุกพล่าน ซึ่งบางครั้งก็เกิดในสถานศึกษาได้โดยเฉพาะเมื่อนักศึกษาเดินพาเหรดเข้าและออกจากห้องเรียน

แต่ผมเดาผิดถนัด เพราะมอดส์ นีการ์ดเป็นหนุ่มไวกิ้งที่แอกทีฟ กระตือรือร้น เสียงดัง และมีความมั่นใจในตัวเองมาก โดยเฉพาะเวลาสอนวิชาที่ได้รับมอบหมาย แม้เขาจะยังไม่ใช่นักคอมพิวเตอร์มีชื่อเสียงระดับโลกเท่ากับหลายๆ คนที่ผมเคยเขียนถึงมาแล้ว แต่ก็น่าสนใจหากได้ลองพูดคุยรับฟังแนวคิดจากเขาโดยเฉพาะทางด้าน Distributed Database ที่เขาชำนาญ

"มอดส์ คุณบอกผมหน่อยซิว่าระบบ Distributed Database หรือ DDB นี่คืออะไร" ผมยิงคำถามใส่เขาในห้องทำงานซึ่งเงียบสงบตามที่เขาต้องการ นั่นไม่ใช่เพราะเรารุ่มรวยห้องที่จะเลือกห้องเงียบๆ ให้เขาได้ดอกครับ แต่เป็นห้องเดียวที่เรามีอยู่หลังจากอาจารย์คนก่อนย้ายออกไป และห้องของเรานั้นเงียบทุกห้อง ถ้าหากไม่มีใครทำเสียงดังรบกวน

"เอาสั้นๆ นะครับ **เจ้า DDB นี้ก็หมายถึงกลุ่มของระบบฐานข้อมูลชนิด centralized ที่กระจายอยู่ตามที่ต่างๆ และเชื่อมโยงถึงกันด้วยระบบสื่อสาร โดยที่แต่ละแห่งนั้นมีข้อมูลของระบบๆ เดียวกระจายอยู่ในคอมพิวเตอร์หลายๆ เครื่อง"** มอดส์ตอบ

"คุณหมายความอย่างไรที่ว่าแต่ละแห่งมีข้อมูลของระบบๆ เดียว" ผมถามต่อ

ผมหมายความว่าคุณมีข้อมูลเรื่องเดียวกระจายอยู่หลายแห่ง ยกตัวอย่างง่าย ๆ เช่น ถ้าคุณมีฐานข้อมูล A อยู่ในคอมพิวเตอร์เครื่องที่ 1 ฐานข้อมูล B อยู่ในคอมพิวเตอร์เครื่องที่ 2 ฐานข้อมูล C อยู่ในคอมพิวเตอร์เครื่องที่ 3 และถ้า A, B และ C เป็นฐานข้อมูลคนละระบบ แม้จะมีระบบสื่อสารโยงถึงกันก็ไม่ใช่ DDB แต่ถ้าฐานข้อมูล A, B และ C เป็นฐานข้อมูลระบบเดียวกันอย่างนี้ก็เป็น DDB ได้ แต่ที่พูดมานี้ยังไม่หมดนะครับ ต้องดูการเรียกค้นหาข้อมูลด้วย" มอดส์อธิบายยิ้ม ๆ แล้วกล่าวต่อว่า "คือ DDB นั้นยอมให้คอมพิวเตอร์เครื่องหนึ่งเรียกใช้ข้อมูลในฐานข้อมูลของคอมพิวเตอร์หลาย ๆ เครื่องได้ หรือในทางกลับกันคอมพิวเตอร์หลาย ๆ เครื่อง อาจเรียกค้นหาข้อมูลในฐานข้อมูลเดียวกันได้"

"ผมฟังแล้วก็ยังงง ๆ อยู่ คุณลองหาตัวอย่างที่เป็นของจริงหน่อยซีครับ" ผมเรียกร้อง

"โอเค ผมคิดว่าธนาคารหลายแห่งในเมืองไทยคงจะมีระบบที่เชื่อมโยงกันทั่วประเทศ ใช่ไหมครับ" มอดส์ถามกลับ

"ใช่ครับ ธนาคารส่วนใหญ่มีคอมพิวเตอร์ระดับเมนเฟรมในกรุงเทพฯ และบางแห่งมีคอมพิวเตอร์ขนาดเล็กกระจายอยู่ที่เชียงใหม่ หาดใหญ่ และที่อื่น ใช้ระบบสื่อสารเชื่อมโยงกับเมนเฟรมในกรุงเทพฯ" ผมตอบ

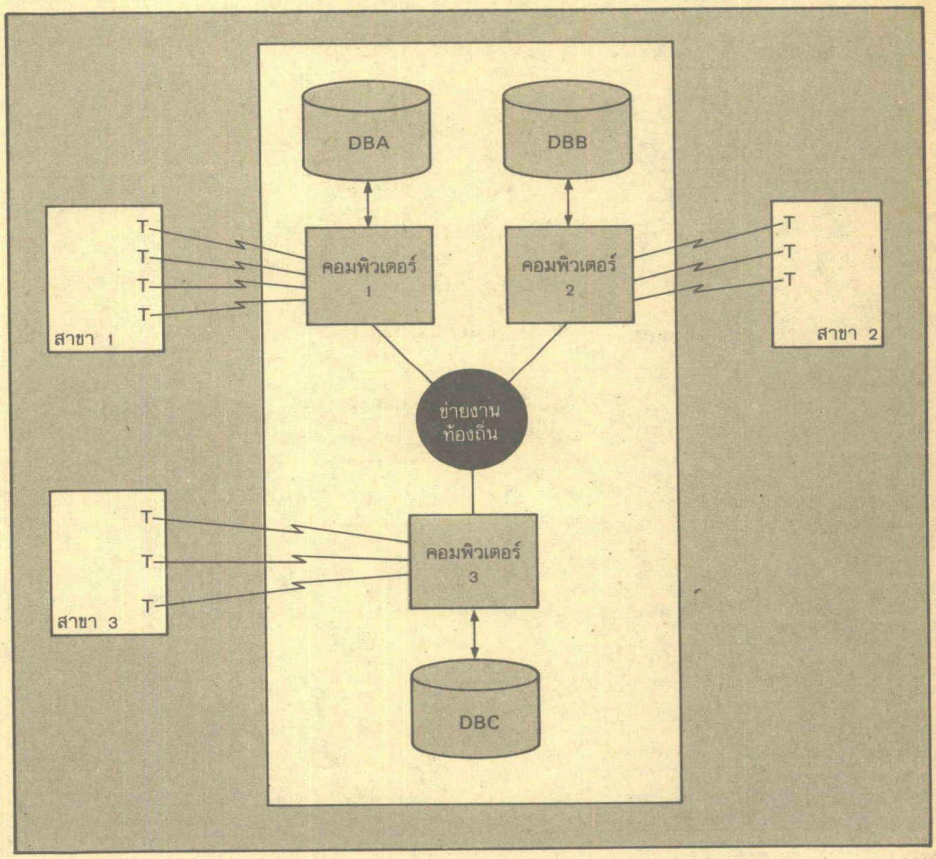
"สมมติว่า ธนาคารแห่งหนึ่งในกรุงเทพฯเก็บข้อมูลของลูกค้าที่กรุงเทพฯ เอาไว้ในเครื่องเมนเฟรม และเก็บข้อมูลลูกค้าที่เชียงใหม่เอาไว้ในคอมพิวเตอร์ของเชียงใหม่ แล้วลูกค้ามาถอนเงิน ลูกค้ากรุงเทพฯก็ถอนเงินที่กรุงเทพฯ ลูกค้าที่เชียงใหม่ก็ถอนที่เชียงใหม่ อย่างนี้จะเป็น DDB ไม่ได้ แม้ว่าคอมพิวเตอร์ที่กรุงเทพฯจะเชื่อมโยงต่อกับคอมพิวเตอร์ที่เชียงใหม่ งานฐานข้อมูลแบบนี้เป็นงานแบบ Local processing"

"ใช่ครับ เพราะไม่มีอะไรเกี่ยวกันเลยระหว่างลูกค้ากรุงเทพฯกับลูกค้าที่เชียงใหม่ แบบนี้ผมพอจะมองเห็น แล้วทีนี้ถ้าลูกค้าที่เชียงใหม่เดินทางมากรุงเทพฯ แล้วใช้บัตรเอทีเอ็มเบิกเงินล่ะครับ"

"อา...คราวนี้เราก็ได้กรณีที่น่าสนใจขึ้น" มอดส์พยักหน้า "เมื่อลูกค้าใช้บัตรเอทีเอ็มที่กรุงเทพฯก็เท่ากับกำลังใช้เครื่องคอมพิวเตอร์ของกรุงเทพฯ ซึ่งผมคิดว่าเครื่องที่กรุงเทพฯจะส่ง Request ต่อไปให้คอมพิวเตอร์ที่เชียงใหม่ค้นหาระเบียนลูกค้า แล้วตอบกลับมาว่าโอเคหรือไม่ งานในลักษณะนี้ผมยังถือว่าเป็น Local processing อยู่ เพราะคอมพิวเตอร์ที่กรุงเทพฯไม่ได้ทำอะไรกับฐานข้อมูลที่เชียงใหม่เลย"

"อ้าว ถ้าอย่างงั้นมันก็ขัดกับที่คุณอธิบายเกี่ยวกับ DDB น่ะซีครับ" ผมท้วง

"ไม่ขัดซีครับ คือ DDB นั้นต้องดูที่การ process ด้วย ในกรณีของข้อมูลธนาคาร แม้เป็นระบบเดียวกัน มีลักษณะสัมพันธ์กันในเชิงตรรกะ แต่งานเท่าที่หยิบมาอธิบายนี้คอมพิวเตอร์เครื่องหนึ่งไม่ได้ process ข้อมูลในแบบสัมพันธ์กันกับคอมพิวเตอร์อื่นเลย คราวนี้ถ้าหากเปลี่ยนการประมวลข้อมูลไปบ้าง เช่น ลูกค้าคนหนึ่งสั่งโอนเงินจากบัญชีที่กรุงเทพฯไปเข้าบัญชีที่เชียงใหม่ คุณจะเห็นว่าคำสั่งชุดเดียวระบุให้มีการ update ฐานข้อมูลถึงสองแห่ง ซึ่งคอมพิวเตอร์ทั้งสองต้องทำงานประสานกันอย่างดี ถ้าคอมพิวเตอร์ที่กรุงเทพฯเริ่มลดยอดเงินในบัญชีที่กรุงเทพฯลง ก็จะต้องแน่ใจว่าคอมพิวเตอร์ที่เชียงใหม่จะต้องเพิ่มยอดเงินในบัญชีได้รับโอน ถ้า update



รูปที่ 2 ฐานข้อมูลแบบกระจายชนิดรวมอยู่ด้วยกันในระบบข่ายงานท้องถิ่น

ต้องทำทั้งสองแห่ง หรือถ้าเกิดเหตุการณ์ขัดข้องก็ต้องไม่ update เลยสักแห่งเดียว ถ้า update เพียงที่ใดที่หนึ่งก็ต้องถือว่าเกิดความผิดพลาด"

"เมื่อเป็นเช่นนี้ผมก็ขอเดาว่าเราจะต้องมีซอฟต์แวร์สำหรับทำงานควบคุม DDB ใช่ไหมครับ"

"ใช่ซีครับ **เราต้องมีซอฟต์แวร์ที่เรียกว่า Distributed DataBase Management System ซึ่งก็เป็นในทำนองเดียวกับ DBMS ของระบบฐานข้อมูลธรรมดา**"

"ซอฟต์แวร์ DDBMS ตัวนี้มีโครงสร้างอย่างไรครับ" ผมถาม

"โอ้โฮ ถามอย่างนี้มันตอบยากนะครับ" มอดส์บ่น "เอายังงี้แล้วกัน ผมมีแผ่นใสแสดง software architecture ของเจ้า DDBMS อยู่แผ่นหนึ่ง รอเดี๋ยวนะ" เขาก้มลงหยิบแฟ้มแผ่นใสขึ้นมาจากลิ้นชักโต๊ะ แล้วก็พลิกหาแผ่นใสที่ต้องการมายื่นให้ผมดู ดังปรากฏในรูปที่ 3

"แล้วนี่มันอะไรบ้างล่ะครับ รู้สึกจะมีส่วนต่าง ๆ หลายส่วนหลายระดับเหลือเกิน"

"ก็มันทำงานที่ค่อนข้างยากมาก ๆ นี่ครับ ก็ต้องมีหลายระดับหน่อย" มอดส์ตอบ

"ไหนลองช่วยอธิบายย่อ ๆ สักหน่อยซีครับ" ผมรบเร้า "เริ่มจากระดับบนสุดเป็นไง รู้สึกจะง่ายกว่าเพื่อน"

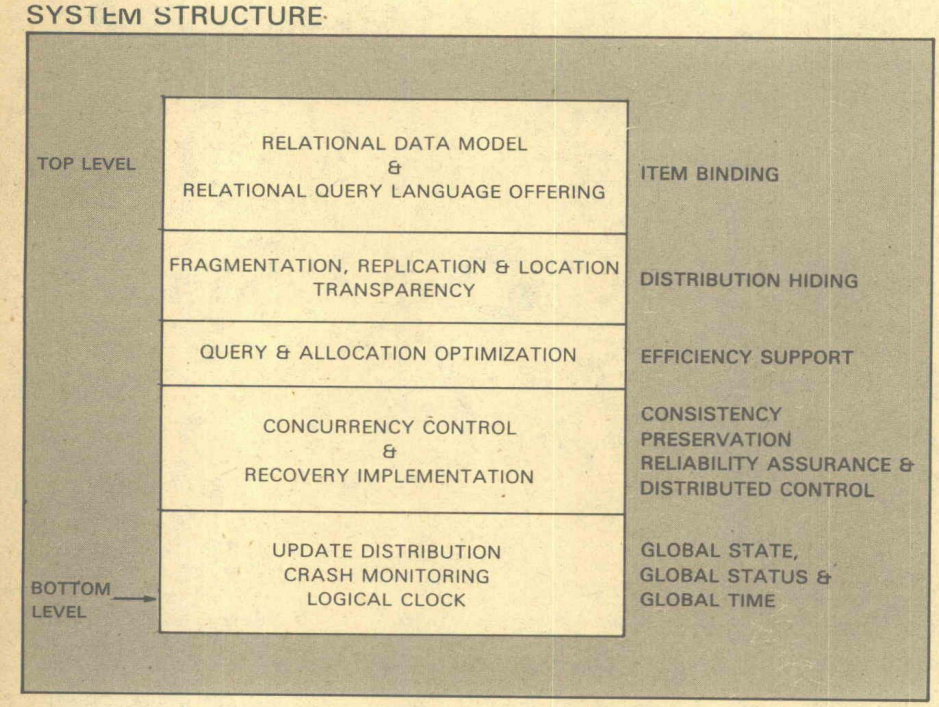
"ระดับบนสุดเราเรียกว่าเป็น Item binding ก็มีหน้าที่เหมือน centralized database management system ธรรมดาคือ มีโมเดลของข้อมูลซึ่งผมสมมุติว่าเป็นชนิด relational แต่คุณจะใช้โมเดลอื่น ๆ ก็ได้นะแล้วแต่ถนัด ระดับนี้เป็นระดับของภาษาสำหรับเรียกค้นหาข้อมูลด้วย...."

"ทำไมถึงเรียกระดับนี้ว่าเป็น item binding ล่ะครับ" ผมซักขึ้นกลางคัน

"ก็มันเป็นระดับชั้นที่นำเอาข้อมูลต่าง ๆ ที่เรียกค้นได้มารวมกันเพื่อส่งให้ผู้ใช้น่ะซีครับ..." มอดส์ตอบ และเมื่อเขาเห็นผมพยักหน้ารับเขาก็อธิบายต่อว่า "ระดับต่อไปเป็นระดับ Distribution Hiding ซึ่งหมายความว่าโปรแกรมที่เรียกใช้ข้อมูลไม่จำเป็นจะต้องรู้ว่าข้อมูลกระจัดกระจายอยู่ที่ไหนบ้าง สำหรับผู้ใช้แล้วโปรแกรมสำหรับ DB กับ DDB นั้นจะไม่แตกต่างกันเลย ถึงตรงนี้ผมอยากจะเพิ่มเติมสักหน่อยว่า ตามหลักการฐานข้อมูลนั้น เราต้องการเก็บข้อมูลชุดเดียวเพื่อหลีกเลี่ยงปัญหาข้อมูลซ้ำหรือ Redundancy ทั้งยังช่วยทางด้าน consistency ของข้อมูลด้วย แต่เมื่อเราเล่นกับ DDB ถ้าเรายึดตามนี้ เราก็อาจแบ่งข้อมูลทั้งระบบเป็นส่วน ๆ ส่งให้คอมพิวเตอร์แต่ละระบบเก็บโดยไม่ให้มีข้อมูลซ้ำกันเลยก็ได้ เรียกว่าเป็นการ Fragmentation แต่ถ้าคุณคิดให้ลึกคุณอาจมีปัญหาว่าจะแบ่งข้อมูลอย่างไรอีกเช่น คุณมีคอมพิวเตอร์สามเครื่องและฐานข้อมูล DDB ของคุณมีตารางสามตาราง คุณอาจจะแบ่งฐานข้อมูลเก็บไว้ในคอมพิวเตอร์เครื่องละตารางก็ได้ หรือคุณจะแบ่งแต่ละตารางเป็นสามส่วนแล้วเก็บหนึ่งในสามของแต่ละตารางลงในแต่ละคอมพิวเตอร์ก็ได้ และที่ไม่คล้องตามหลักการ DB เลยก็คือ เก็บทั้งสามตารางนั่นแหละไว้ในแต่ละคอมพิวเตอร์ ถึงมันจะมีข้อมูลซ้ำและเปลืองที่เก็บสักหน่อยก็ไม่เป็นไร"

"ผมคิดว่าภารเลือกว่าจะแบ่งเก็บแบบไหนคงแล้วแต่ Performance ที่ต้องการถูกไหมครับ" ผมปรารภ พร้อมกับเริ่มรู้สึกฉงนฉงายในความซับซ้อนของเจ้า DDB นี้

"จริงครับ แต่ยังมีองค์ประกอบอื่น ๆ อีก เช่น ความมั่นคงปลอดภัยของข้อมูล ความสม่ำเสมอของระบบสื่อสารข้อมูล คราวนี้เราลองดูระดับที่สามนะครับ ระดับนี้เป็นระดับ Efficiency Support ซึ่งมุ่งหมายที่จะช่วย



รูปที่ 3 สถาปัตยกรรมซอฟต์แวร์ของฐานข้อมูลแบบกระจาย

ให้งานเรียกค้นและใช้ข้อมูลใน DDB มีประสิทธิภาพขึ้น ในระดับนี้ก็มี Query Optimization ซึ่งใช้หลักการคณิตศาสตร์กำหนดว่าจะเรียกค้นข้อมูลในแบบใดจึงจะดีที่สุด แล้วก็มี Allocation Optimization ซึ่งกำหนดว่าจะแบ่งย่อยข้อมูลไปเก็บในคอมพิวเตอร์ใดบ้าง"

"เอ๊ะ...เดี๋ยวก่อนครับ" ผมขัดจังหวะเพราะรู้สึกมีอะไรแปร่ง ๆ "เมื่อกี้คุณพูดถึง Fragmentation ซึ่งก็เกี่ยวกับการแบ่งข้อมูลรวมเป็นข้อมูลย่อย แล้วคราวนี้มาพูดเรื่อง Allocation ซึ่งคุณบอกว่าเป็นการเก็บข้อมูลย่อยไปไว้ในคอมพิวเตอร์ต่างๆ ผมว่ามันเป็นเรื่องเดียวกันนะครับ"

"ไม่ใช่เรื่องเดียวกันครับ" มอดส์ตอบ "หลักการ Fragmentation นั้นใช้กับกรณีที่ผู้ใช้ฐานข้อมูลทำงานกับข้อมูลรวมในลักษณะ Global ส่วนหลักการ Allocation นั้นใช้ในลักษณะ local คือใช้ข้อมูลย่อย โดยที่ทั้งสองกรณีผู้ใช้ไม่รู้และไม่จำเป็นต้องรู้ว่า ตัวจริงของข้อมูลนั้นอยู่ที่ไหน พูดง่ายๆ fragmentation กล่าวว่าจะแบ่งข้อมูลเป็นส่วนๆ ได้อย่างไร และ allocation กล่าวว่าจะเก็บข้อมูลไว้ที่ไหน"

"เอาละครับ ผมพอจะมองเห็นแนวคิดบ้างแล้ว" ผมยอมรับหลังจากนั่งคิดดูสักครู่

"ระดับที่สี่เป็นระดับที่เกี่ยวกับการควบคุมและป้องกันให้ข้อมูลมีความถูกต้องเชื่อถือได้ ซึ่งประกอบด้วยงาน Concurrency Control และ Recovery" มอดส์อธิบายต่อไป "Concurrency Control นั้นควบคุมการที่ผู้ใช้หลาย ๆ คนต้องการเรียกใช้ข้อมูลชุดหนึ่งในเวลาเดียวกัน เช่น ผู้ใช้ A อ่านค่าในระเบียน X เสร็จแล้วก็จะ update ค่าของ X แต่ระหว่างนั้นผู้ใช้อีกรายคือ B ก็ต้องการอ่านค่าในระเบียน X เหมือนกัน ถ้า B อ่านค่า X หลังจากที่ A update ค่า X เสร็จไปแล้วก็ไม่เป็นไร

รูปที่ 4 ระบบ ATM ของธนาคารบางแห่งมีการกระจายข้อมูลไว้ในคอมพิวเตอร์หลายระบบ

แต่ถ้า B อ่านหลังจาก A อ่านเสร็จแต่ยังไม่ทัน update ก็อาจมีปัญหาได้...."

"แต่ B ก็ได้ค่าของ X ไปนะครับ ทำไมถึงจะมีปัญหา" ผมขัดจังหวะอีก

"คุณลองคิดถึงกรณีของธนาคารนะ สมมุติว่า A กำลังจะเอาเงินเข้าบัญชี ดังนั้นจึงต้องอ่านค่ายอดเงินในบัญชีก่อน แล้วจึงบวกด้วยเงินที่จะนำเข้าบัญชี เสร็จแล้วก็บันทึกเก็บ คราวนี้ B เอาเช็คมาขึ้นเงิน ถ้า B นำเช็คมาขึ้นเงินหลังจาก A บันทึกยอดเงินค่าใหม่เข้าไปแล้วก็ไม่เป็นไร แต่ถ้า B อ่านได้จากยอดเงินระหว่างที่ A กำลัง Update แต่ยังไม่เสร็จ B ก็จะได้รับคำตอบว่ามีเงินไม่พอจ่าย ซึ่งเสียหายมากใช่ไหม"

"ครับ ผมพอจะมองออกแล้ว"

"ดีครับ เจ้า concurrency control นี้ลำพังเป็น centralized database ก็ยุ่งพอดูอยู่แล้ว แต่เมื่อเป็น DDB มันยิ่งยุ่งยากขึ้นไปใหญ่ซึ่งผมจะไม่กล่าวถึงละครับ ถัดจากนั้นก็มี Recovery ซึ่งสำคัญมาก เพราะถ้าคอมพิวเตอร์บางเครื่องไม่ทำงาน การบันทึกข้อมูลก็จะผิดพลาด และต้องมีการกู้ระบบหรือถ้าฐานข้อมูลเสียหายไปด้วยประการใด ๆ ก็ต้องมีการกู้ระบบเช่นกัน" มอดส์อธิบาย

"แล้วคราวนี้เราก็มาถึงระดับล่างสุด" ผมเอ่ยขึ้นหลังจากพิจารณารูปข้างต้น

"ครับ ระดับนี้เป็นการควบคุมสภาวะสถานการณ์และเวลาของทั้งระบบ ควบคุมการ update ของแต่ละเครื่องว่าเป็นไปตามเงื่อนไขและการกำหนดทางด้าน fragmentation และ allocation แม้แต่เวลาก็เช่นกัน ต้องมีการ synchronize กันอย่างดีภายในระบบ จะเชื่อถือเวลา Physical time ของคอมพิวเตอร์แต่ละเครื่องไม่ได้"

"จริงครับ เหมือนอย่างภาพยนตร์สายลับ ก่อนปฏิบัติการใด ๆ ต้องเทียบเวลาให้ตรงกันเสียก่อน" ผมกล่าวเสริม แล้วถามต่อว่า "ในบรรดารายละเอียดต่าง ๆ เหล่านี้คุณคิดว่าอะไรที่สำคัญที่สุด หรืออะไรเป็นปัญหาสำคัญที่สุดของระบบ DDB ครับ"

"ถ้าพูดกันถึงความสำคัญ ในฐานะที่มันเป็นโครงสร้างของซอฟต์แวร์ทั้งหมด ผมเห็นว่ามันมีความสำคัญเท่าเทียมกัน จะขาดส่วนหนึ่งส่วนใดไปก็ไม่ได้" มอดส์ตอบยิ้ม ๆ กับคำถามที่ไม่ค่อยจะเข้าท่าของผม "แต่ถ้าพูดถึงปัญหาสำคัญของการใช้ DDB ละก็ผมคิดว่าความมั่นคงปลอดภัยหรือ security เป็นเรื่องสำคัญมากเพราะ DDB ต้องอาศัยระบบสื่อสารข้อมูลซึ่งมีสิทธิ์ถูก tap หรือดักสัญญาณได้ง่าย อย่างในอเมริกาคุณก็เห็นแล้วว่าพวก Hacker สามารถเจาะเข้าไปอ่านความลับในคอมพิวเตอร์ของ FBI และของ CIA ได้ ปัญหาข้อที่สองก็คือ bug ผมแน่ใจว่าระบบ DDBMS ใหญ่ ๆ นั้นยังมี bug อยู่อีกมาก แล้วก็ไม่รู้ว่าเมื่อใดมันจะแผลงฤทธิ์เล่นงานเราให้เจ็บปวด ส่วนปัญหาข้อที่สามก็คือค่าใช้จ่าย แม้ว่าระบบสื่อสารจะสะดวกสบายและมี

รูปที่ 5 งานข้อมูลทะเบียนมีลักษณะกระจายไปทั่วประเทศ

ประสิทธิภาพ แต่มันไม่ใช่บริการให้เปล่านะครับ มันต้องเสียเงินและพอคุณใช้ไปเพลิน ๆ คุณก็จะต้องตกใจที่ค่าใช้จ่ายมันมากเหลือเกิน"

เรื่องนี้ผมเห็นด้วยครับ เวลานี้โทรศัพท์เมืองไทยยังคิดค่าบริการครั้งละ 3 บาทอยู่ ไม่ว่าจะพูดนานแค่ไหน ต่อไปปีหน้าเขาจะเก็บเงินตามเวลาที่ใช้ ดังนั้นใครที่กำลังเล่น Bulletin Board หรือสื่อสารข้อมูลถึงกันอยู่คงต้องปวดหัวใจแน่ ๆ คราวนี้ผมอยากถามต่อว่า DDB นั้น อาจจะกระจายอยู่ห่างกันหลายจังหวัด แบบนี้เราจะแน่ใจได้อย่างไรว่าข้อมูลที่เก็บในฐานข้อมูลแต่ละแห่งนั้นจะถูกต้องเป็นปัจจุบันตลอดเวลา"

"คุณไม่มีทางแน่ใจได้เลย" มอดส์ตอบกลับมาในทันทีทันใด "การที่ข้อมูลในฐานข้อมูลต่าง ๆ จะถูกต้องเป็นปัจจุบันหรือไม่นั้นขึ้นอยู่กับการ update ข้อมูล และการ update ข้อมูลก็จะขึ้นอยู่กับการ Algorithm ที่คุณใช้ ผมจะยกตัวอย่างนะครับ สมมุติว่าคุณใช้วิธี write all หมายถึงให้ update ข้อมูลเดียวกันทุกชุดไม่ว่าจะมีซ้ำอยู่ในคอมพิวเตอร์ใด ๆ ถ้าหากคอมพิวเตอร์ทุกเครื่องในระบบทำงานได้ดีไม่มีการขัดข้องมันก็จะปรับปรุงข้อมูลให้ได้ตามต้องการ และข้อมูลของคุณในคอมพิวเตอร์ทุกแห่งก็จะเป็นปัจจุบันเสมอ แต่ถ้าหากคอมพิวเตอร์เครื่องใดเครื่องหนึ่งขัดข้องระบบก็จะไม่ปรับปรุงข้อมูลเลยสักแห่งเดียว เพราะถ้าปรับปรุงบ้างไม่ปรับปรุงบ้าง ข้อมูลในฐานข้อมูลก็จะต่างกัน..."

"ผมเชื่อว่าการปรับปรุงคงไม่ทำพร้อมกันในวินาทีเดียวกัน ดังนั้นถ้าหากเรามีคอมพิวเตอร์อยู่หลายเครื่องแล้วระบบมัน update ไปแล้วบางเครื่องก็ยังไม่ครบ พอดีไปพบว่าคอมพิวเตอร์เครื่องหนึ่งเกิด down ขึ้นมาอย่างนี้ระบบจะทำอย่างไรครับ" ผมถาม

"อย่างนี้มันก็ทำต่อไม่ได้นั่นหนึ่งละ จากนั้นมันก็ต้องเลิกการ Update ที่ทำไปแล้วทั้งหมดสำหรับคำสั่งนั้น คือมันต้องมี log ที่จะเก็บรายละเอียดว่าทำอะไรบ้าง แล้วนำ log นี้มาประกอบในการยกเลิกหรือ Undo

"ดังนั้นเราก็จะแน่ใจว่าข้อมูลต้องไม่ลักหลั่นกัน ใช่ไหมครับ" ผมเปรยขึ้น

"ใช่ครับ" มอดส์ตอบรับ "แต่วิธีนี้มีปัญหาอย่างที่เห็น ๆ เพราะถ้าคอมพิวเตอร์เสียไปสักเครื่องก็ไม่ต้องทำอะไรกันเลย ดังนั้นจึงมีวิธีอื่น ๆ อีกเช่น write all available คือ update ฐานข้อมูลของคอมพิวเตอร์ทุกเครื่องที่ใช้งานได้ ส่วนเครื่องที่ down นั้นพอใช้ได้แล้วค่อย update ตามหรือ write master คือกำหนดเครื่องให้เป็น master แล้ว update เฉพาะเครื่องนี้ก่อน"

"แต่สองวิธีนี้คุณจะไม่ได้ข้อมูลที่ถูกต้องเสมอไปนะครับ" ผมท้วง

"ใช่ครับ มันต้องแล้วแต่งาน บางงานคุณไม่แคร์นี่ครับว่าข้อมูลจะต้องเป็นปัจจุบันถึงวินาทีนี้" มอดส์ตอบ

"ฟังดูแล้วรู้สึกยุ่งยากจังครับ" ผมบ่น "ถ้ามันมีอะไร ๆ ต้องคำนึงถึงมากมายอย่างนี้ การออกแบบฐานข้อมูลชนิด DDB นี่จะทำอย่างไรล่ะครับ"

"คำถามนี้เป็นคำถามที่ผมตอบไม่ได้แน่ ๆ ครับ เพราะผมเองก็เล่นทางทฤษฎีมากกว่าใช้งานจริง" มอดส์สารภาพ "แต่คำถามเรื่องนี้มีอยู่ในตำราชื่อ Distributed Database ของ Stefano Ceri พิมพ์โดยบริษัท McGraw-Hill ปี 1984 นี้แหละครับ คุณคงจะพอหามาอ่านได้"

นั่นก็เป็นคำแนะนำที่ดีสำหรับผู้สนใจศึกษาเรื่อง Distribute Database อย่างจริงจังต่อไป ส่วนในบทความนี้ผมคงไม่มีเนื้อที่พอจะขยายความเรื่องนี้ได้อีกแล้ว จึงขอจบการสนทนากับหนุ่มไวกิ้งไว้เพียงแค่นี้ อนึ่ง ผมเข้าใจว่าบทสนทนาตอนนี้ค่อนข้างจะรุงรังไปด้วยภาษาอังกฤษ ซึ่งก็จำเป็นเพราะเป็นเนื้อหาที่ยากขึ้นไปอีกระดับหนึ่ง จึงยังไม่มีใครคิดบัญญัติศัพท์ไทย ๆ ให้ใช้ ถ้าผมจะคิดขึ้นใช้เองเดี๋ยวท่านผู้อ่านก็จะต้องแปลไทยกลับเป็นอังกฤษใหม่ก็จะยิ่งยากเกินกว่าเหตุ อย่างไรก็ตามถ้าท่านผู้อ่านนึกคำไหนดี ๆ เพราะ ๆ สำหรับศัพท์ต่อไปนี้ เช่น information hiding, fragmentation concurrency control ได้แล้วละก็อย่าลืมบอกมาให้ทราบบ้างก็แล้วกันครับ